

REF AL

(19) 日本国特許庁 (JP)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平10-91448

(43) 公開日 平成10年(1998) 4月10日

Int Cl⁸

G 0 6 F 9/44

識別記号

5 3 0

FI

G 0 6 F 9/44

5 3 0 P

5 3 0 M

審査請求 未請求 請求項の数18 OL (全 9 頁)

(21) 出願番号 特願平9-178320

(22) 出願日 平成9年(1997) 7月3日

(31) 優先権主張番号 08/675846

(32) 優先日 1996年7月3日

(33) 優先権主張国 米国 (US)

(71) 出願人 390039413

シーメンス アクチエンゲゼルシャフト

SIEMENS AKTIENGESEL

LSCHAFT

ドイツ連邦共和国 ベルリン 及び ミュ

ンヘン (番地なし)

(72) 発明者 ハンス-エーリッヒ ラインフェルダー

ドイツ連邦共和国 エアランゲン プラー

テンシュトラッセ 23

(72) 発明者 カールハインツ ドルン

ドイツ連邦共和国 カルヒロイト エアレ

ンシュトラッセ 29

(74) 代理人 弁理士 矢野 敏雄

(外1名)

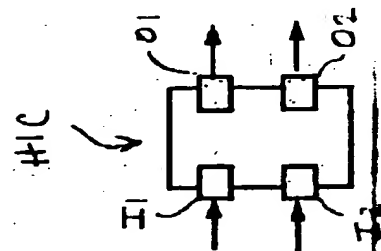
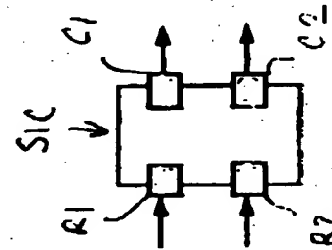
最終頁に続く

(54) 【発明の名称】 オブジェクト指向コンピューティングシステム

(57) 【要約】

【課題】 大きなアプリケーションへの、意味構造のないソフトウェアコンポーネントまたはビルディングブロックを、ビルディングブロック内でのコード変化なしで、そしてアダプタを書くことなしに独立して結合するための方法および/または手段を提供する。

【解決手段】 意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有するようにする。



BEST AVAILABLE COPY

【特許請求の範囲】

【請求項1】 意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とするオブジェクト指向コンピューティングシステム。

【請求項2】 オブジェクトの入力と出力はそれぞれ、CsaConnectableとCsaRemoteオブジェクトを介して供給される、請求項1記載のオブジェクト指向コンピューティングシステム。

【請求項3】 入力と出力に関連した各データ構造が別個のヘッダファイルに記述されており、該ヘッダファイルはリンクすべき各オブジェクトにより使用される、請求項2記載のオブジェクト指向コンピューティングシステム。

【請求項4】 各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力と出力のASCII構築ファイル名のランタイムでダイナミックにリンク可能である、請求項2記載のオブジェクト指向コンピューティングシステム。

【請求項5】 ダイナミックにリンク可能な入力および出力と、オブジェクトへおよびオブジェクトからそれぞれ前記入力および出力を介して伝送されるデータをキューイングするための内部タスクと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とするオブジェクト指向コンピューティングシステム。

【請求項6】 オブジェクトの入力と出力は、CsaConnectableオブジェクトとCsaRemoteオブジェクトをそれぞれ介して供給される、請求項5記載のオブジェクト指向コンピューティングシステム。

【請求項7】 入力および出力に関連する各データ構造が別個のヘッダファイルに記述されており、該ヘッダファイルはリンクすべき各オブジェクトにより使用される、請求項6記載のオブジェクト指向コンピューティングシステム。

【請求項8】 各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力と出力の名前を含むASCII構築ファイルによるランタイムでダイナミックにリンク可能である、請求項6記載のオブジェクト指向コンピューティングシステム。

【請求項9】 オブジェクト指向コンピューティングシステムに含まれるソフトウェアコンポーネントを設計するための方法において、

十分に分散可能な入力と出力を定義し、リンク可能であり、意味構造のないソフトウェアコンポーネントを入力および出力接続点によって構築し、十分に分散可能なイベントに基づくオートルーティンされるパターンを、イベント通信フレームワークに基づ

いて供給する、ことを特徴とする方法。

【請求項10】 コンピュータプラットフォーム上にオブジェクト指向コンピューティングシステムを有するオブジェクト指向コードを含む記憶媒体において、意味構造のない、ダイナミックにリンク可能な入力および出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とする記憶媒体。

【請求項11】 オブジェクトの入力と出力は、CsaConnectableオブジェクトとCsaRemoteオブジェクトを介してそれぞれ供給される、請求項10記載の記憶媒体。

【請求項12】 入力および出力に関連する各データ構造は別個のヘッダファイルに記述されており、該記述ファイルはリンクすべき各オブジェクトにより使用される、請求項11記載の記憶媒体。

【請求項13】 各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力および出力の名前でファイルされたASCIIコンフィギュレーションによってランタイムでダイナミックにリンク可能である、請求項11記載の記憶媒体。

【請求項14】 コンピューティングシステム上のオブジェクト指向コンピューティングシステムに対するオブジェクト指向コードを含む記憶媒体において、ダイナミックにリンク可能な入力および出力と、オブジェクトへおよびオブジェクトから前記入力および出力をそれぞれ介して伝送されるデータのキューイングのため内部タスクと、

自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有する、ことを特徴とする記憶媒体。

【請求項15】 オブジェクトの入力および出力は、CsaConnectableオブジェクトおよびCsaRemoteオブジェクトを介してそれぞれ供給される、請求項14記載の記憶媒体。

【請求項16】 入力および出力に関連する各データ構造は別個のヘッダファイルに記述されており、該ヘッダファイルはリンクすべき各オブジェクトにより使用される、請求項15記載の記憶媒体。

【請求項17】 各オブジェクトは共有ライブラリであり、該共有ライブラリは、オブジェクトの入力および出力の名前を含むASCII構築ファイルにより、ランタイムでダイナミックにリンク可能である、請求項15記載の記憶媒体。

【請求項18】 オブジェクト指向コンピューティングシステムでソフトウェアコンポーネントを設計するための方法に対するオブジェクト指向コードを含む記憶媒体において、

十分に分散可能な入力および出力イベントを定義し、ダイナミックにリンク可能で、意味構造のないソフトウ

ェアコンポーネントを入力および出力接続点により構築し、十分に分散可能なイベントに基づいてオートルーティングされるパターンをイベント通信フレームワークに基づいて供給する、ことを特徴とする記憶媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、オブジェクト指向マルチプログラミングシステムに関する。より具体的には、本発明は、ソフトウェアコンポーネントまたはビルディングブロックを相互接続するための方法および手段に関するものである。

【0002】

【従来の技術】米国特許第5499365号明細書に記載されているように、オブジェクト指向プログラミングシステムおよびプロセス（“オブジェクト指向コンピュータ環境”とも称される）は、多くの研究と興味の対象である。当業者には周知であるように、オブジェクト指向プログラミングシステムは多数の“オブジェクト”から構成される。1つのオブジェクトはデータ構造であり、“フレーム”と称される。演算または関数のセットはまた“メソッド”とも称され、データ構造にアクセスすることができる。フレームは“スロット”を有することができる。各スロットは、スロット内のデータの“属性”を含む。この属性は（整数またはストリングのような）基関数、または他のオブジェクトに対するポインタであるオブジェクトレファレンスとすることができる。識別データ構造と共通の特性を有するオブジェクトは相互に群に入れることができ、1つの“クラス”として正しく識別される。

【0003】オブジェクトの各所定のクラスは通常、“インスタンス”の数に現れる。各インスタンスは所定のデータ構造をオブジェクトの所定の例に対して含む。オブジェクト指向コンピュータ環境ではデータは次のように処理される。すなわち、オブジェクトにそのメソッドの1つを、オブジェクト“メッセージ”を送信することによって実行することを要求することにより処理される。受信オブジェクトはメッセージに次のようにして応答する。すなわち、メッセージネームを満たすメソッドを選択し、このメソッドを名付けられたメソッドに基づいて実行し、コントロールをコーリング・ハイレベルルーティンに、このメソッドの結果に沿って戻すのである。クラス間の関係、オブジェクトおよびインスタンスは伝統的に“形成時間”中、またはオブジェクト指向コンピュータ環境の発生中に確立される。すなわち、“ランタイム”より時間的に前、またはオブジェクト指向コンピュータ環境の実行前に確立される。

【0004】上記のクラス、オブジェクトおよびレファレンス間の関係に加えて、継承関係もまた2つまたはそれ以上のクラス間に存在する。すなわち、第1のクラス

は第2のクラスの“ペARENT”と見なすことができ、第2のクラスは第1のクラスの“チャイルド”と見なすことができる。言い替えると、第1のクラスは第2のクラスの祖先であり、第2のクラスは第1のクラスの子孫である。従って第2のクラス（すなわち子孫）は第1のクラス（すなわち祖先）から引き継がれる。チャイルドクラスのデータ構造はペARENTクラスの属性のすべてを含む。

【0005】オブジェクト指向システムはこれまで、オブジェクトの“バージョン”であると認識されていた。オブジェクトのバージョンは、異なる時点でのオブジェクトで同じデータである。例えば、“ワーク・イン・プログレス”に関連するオブジェクトは完成した立証済のワークに関連する同じオブジェクトとは別個のバージョンである。多くのアプリケーションはまた、データがそれぞれ異なる時点で存在したというデータの歴史的記録を要求する。従ってオブジェクトの別のバージョンが要求される。

【0006】2つの刊行物がさらに一般的背景を示している。E.W.Dijkstra, The Structure of "THE" Multi programming System, Communications of the ACM, Vol. 11, No. 5, May 1968, pp. 341-346, と C.A.R. Hoare, Monitors: Operating Systems Structuring Concepts, Communications of the ACM, Vol. 17, No. 10, October 1974, pp. 549-557である。両方の刊行物は互いを引用している。前者の刊行物は、プリミティブを使用した同期方法について記載し、セマフォの使用を説明するものであり、後者の刊行物はモニタのBrinch-Hansensコンセプトをオペレーティングシステムの構築方法として開発したものである。とりわけ、Hoareの刊行物は、プロセスに対する同期の形態を紹介し、セマフォに関し可能な移植法について述べており、フルフルールについても実例を示している。

【0007】Hoareの刊行物に記載されているように、オペレーティングシステムの第1の目的は、計算機設置をそのリソースに対する要求が干渉不能である多くのプログラム間でシェアすることである。従って設計者の第1の役目は、種々の種類のリソース（例えば主記憶装置、ドラム記憶装置、磁気テープ・ハンドラ、コンソール）に対するスケジュールアルゴリズムによるリソースアロケーションをデザインすることである。この役目を簡単にするため、プログラマはリソースの各クラスごとに別個のスケジューラを構成しようと試みる。この場合各スケジューラは所定量のローカル管理データからなり、いくつかのスケジューラおよびファンクションと共に構成される。これらスケジューラおよびファンクションは、プログラムによりリソースを獲得および解放するために呼び出される。このような関連したデータとプロセスの収集はモニタとして知られる。

【0008】適応通信環境（ACE）は、Douglas C. S

Schmidt (Assistant Professor with the Department of Computer Science, School of Engineering and Applied Science, Washington University) により開発されたネットワークプログラミングシステムのオブジェクト指向タイプである。ACEは、ユーザーレベルユニットとWIN32 (Windows NT および Windows 95) OSメカニズムを、形式安全で、効率的で、オブジェクト指向インターフェースによって1つにまとめる。

【0009】・IPCメカニズム—インターネットドメインとUNIXドメイン・ソケット、TLI、名前付きパイプ (UNIXとWIN32に対し) およびSTREAMパイプ;

・イベントマルチプレクサー—UNIXでselect()とpoll()により、WIN32でWaitForMultipleObjectsにより;

・ソラリススレッド、POSIX Pスレッド、およびWIN32スレッド;

・明示的ダイナミック・リンキング機構—例えばUNIXのdlopen/dlclose/dlclose、WIN32のLoadLibrary/GetProc;

・メモリマップファイル;

・システムV RPC—メモリのシェア、セマフォ、メッセージキュー;

・サンRPC (GNU C++)

付加的に、ACEは多数のハイレベルクラスカテゴリーとネットワークプログラミングフレームワークを、ローレベルC++ラッパーを統合し、強調するために有している。ACEのハイレベルコンポーネントは、コンカレント・ネットワーク・デーモンをサポートする。これらのデーモンはアプリケーションサービスからなる。ACEは最近、多数の商用プロダクトで使用される。この商用プロダクトは、ATMリングナリングソフトウェアプロダクト、I/Oモニタアプリケーション、ネットワークマネージメント、および移動通信システムと事業拡大分散専用システムに対する一般的ゲートウェイ通信を含む。ACEについての重要な情報と文獻は、WWW上で次のURLから得ることができる。

【0010】<http://www.wustl.edu/~schmidt/ACE-overview.html>

以下の省略形が本出願で使用される。

【0011】スレッド—プロセス内での平行実行ユニット。モニタ同期、強制的逐次化により、複数同時実行スレッドの並列アクセス。このスレッドはすべてモニタによりプロテクトされている1つのオブジェクトの機能をコールアップする。

【0012】・同期化—プリミティブ—パラレルアクティビティの反復的止断化に対するオペレーティングシステムの手段

・セマフォ—パラレルアクティビティに対する同期化プリミティブ

・Mutex—パラレルアクティビティに対する専用同期化プリミティブ、相互実行目的のために、これはクリティカルなコードレンジを含む。

【0013】・コンディションキュー—所定の条件を参照するパラレルアクティビティに対するキューを待つイベント

・ゲートロック—各エントリ—ファンクションに対するモニタのMutex、オブジェクトをプロテクションするため、オブジェクトの1つのエントリ—ルーチンを使用するため同時には1つのパラレルアクティビティだけを許容するため。

【0014】・ロングタームスケジュール—コンディションキュー内またはパラレルアクティビティに対するキュー待ちイベント内での1つのパラレルアクティビティの長期遅延。

【0015】・ブローカ—ディストリビュータ付加的に次の省略形が使用される。

【0016】AFM Asynchronous Function Manager (非同期ファンクションマネージャ)

SESAM Service & Event Synchronous Asynchronous Manager (サービスおよびイベント同期非同期マネージャ)

PAL Programmable Area Logic (プログラマブルエリアロジック)

API Application Programmers Interface (アプリケーション・プログラマー・インターフェース)

IDL Interface Definition Language (インターフェース定義言語)

ATOMIC Asynchron Transport Optimizing observer-pattern-like systems supporting several Modes (client/server-push/pull) for an IDL-less Communication subsystem (IDLレス通信システムに対する非同期伝送最適オブザーバパターン様システムサポート複数モード)

XDR External Data Representation

I/O 入力/出力

CSA Common Software Architecture (a Siemens AG computing system convention) (共通ソフトウェアアーキテクチャ) (シーメンス社コンピューティングシステム規定)

SW ソフトウェア (Software)

過去において、ソフトウェアコンポーネントまたはビルディングブロックのインターフェースはアプリケーションプログラムインターフェース (API) にコード化するのが困難であった。この解決手段はプロセスにリンク可能であったが、しかしロケーション透過型ではなかった。さらにインターフェースは、コード化の困難なオブジェクトリファレンスを備えたインターフェース定義言語 (IDL) によって行われていた。

【0017】

【発明が解決しようとする課題】本発明の課題は、大きなアプリケーションへの、意味構造のないソフトウェアコンポーネントまたはビルディングブロックを、ビルディングブロック内でのコード変化なしで、そしてアダプタを書くことなしに独立して結合するための方法および/または手段を提供することである。

【0018】

【課題を解決するための手段】上記課題は本発明により、意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、自動的に、パターンに基づいて、十分に分散可能なイベントを供給するイベント通信フレームワークとを有するように構成して解決される。

【0019】

【発明の実施の形態】そのために、コンポーネントまたはビルディングブロックの機能は周りの環境から完全に分離され、これにより同じプロセス内でロケートする必要がなく、その代わりにネットワークを介して分散することができ、その際にインターフェース定義言語を必要としない。

【0020】実施例では、ソフトウェアコンポーネントを設計するための本発明の方法は次のステップを有する：すなわち、十分に分散可能な入力イベントおよび出力イベントを定義し；ダイナミックにリンク可能であり、意味構造のないソフトウェアコンポーネントを入力および出力接続点により構築し；十分に分散可能なイベントに基づくオートルーティングパターンを、イベント通信フレームワークに基づいて設定する。

【0021】別の実施例では、本発明はオブジェクト指向コンピューティングシステムを提供する。このシステムは、意味構造のない、ダイナミックにリンク可能な入力と出力を備えたオブジェクトと、オートルーティングされ、パターンに基づき、十分に分散可能なイベントを設定するイベント通信フレームワークを有する。

【0022】別の実施例では、オブジェクトの入力と出力はCsaConnectableオブジェクトとCsaRemoteオブジェクトへのリンクによってそれぞれ設定される。

【0023】別の実施例では、入力および出力に関連した各データ構造が別個のヘッダファイルに記述され、このヘッダファイルをリンクすべきすべてのオブジェクトが使用することができる。

【0024】別の実施例では、各オブジェクトは共有ライブラリであり、オブジェクトの入力および出力のASCII構築ファイル名によってランタイムでダイナミックにリンクすることができる。

【0025】

【実施例】本発明を以下、有利な実施例に基づいて詳細に説明する。

【0026】前に述べたように本発明は、意味構造のない

ビルディングブロックを大きなアプリケーションに、ビルディングブロック内でのコード変化なしで、かつアダプタを書くことなしに独立して結合するための手段である。このことにより、ソフトウェアブロックを、集積回路を結合できるのと同じ仕方で結合することのできるシステムまたはアーキテクチャが得られる。

【0027】本発明の目的のために、ソフトウェアビルディングブロックはその外環境（これはユーザーライトコードと同じように、1つ以上の他のビルディングブロックからなることができる）と、CsaConnectable（提供者）およびCsaRemote（消費者）オブジェクト/クラスを介して接続される。これは、他の接続終点が同じプロセスにあるかまたはリモートホスト（ローカルの場合は最適化により）にあるかは関係ない。このルールはすべてのビルディングブロックに適用される。

【0028】ビルディングブロックの入力および出力に関連する各データ構造は別個のヘッダファイルに記述される。このヘッダファイルは接続すべきすべてのビルディングブロックが使用できる。

【0029】各ビルディングブロックは共有ライブラリとして実現または構成される。この共有ライブラリは、パブリックドメイン通信パッケージで、ASCII構築ファイルが使用されるのと同じようにランタイムでダイナミックにリンクされる。入力および出力の名前はダイナミックリンクの間に割り当てられ、これにより再コンパイルまたは再リンクなしでコンフィギュレーションを変化することができる。

【0030】この手段の大きな利点は、フレキシブルであることと、ハイレベルパターンが他のシンプルな、よくデザインされた、意味構造のないパターンの最適な結合から得られることである。さらにこの機構は、ハードウェアの世界のオンボード集積回路の結合と非常によく似ている。

【0031】図1は、ハードウェア集積回路（IC）と本発明のソフトウェアオブジェクトとの類似性を比較するのに有用である。図1では、ハードウェアIC H1Cが2つの入力ピンI1、I2と、2つの出力ピンO1、O2を有する。同じようにソフトウェアオブジェクトS1Cは、CsaRemoteを介する2つの入力R1、R2と、CsaConnectableを介する2つの出力C1、C2を有する。

【0032】このようなソフトウェアICシステムの実施のための符号化の例を以下に示す。

【0033】1. 入力/出力クラス定義

【0034】

【外1】

```

#ifndef SAMPLECLASS1H
#define SAMPLECLASS1H
/*****
 *
 *   Input/Output data structure
 *
 *****/
struct SampleClass1 {
    int         theInteger;
    DECLARE_MSC (SampleClass1)
};
IMPLEMENT_MSC (SampleClass1, V(theInteger))
#endif // SAMPLECLASS1H

#ifndef SAMPLECLASS2H
#define SAMPLECLASS2H
/*****
 *
 *   Input/Output data structure
 *
 *****/
struct SampleClass2 {
    int         theInteger;
    DECLARE_MSC (SampleClass2)
};
IMPLEMENT_MSC (SampleClass2, V(theInteger))
#endif // SAMPLECLASS2H

```

【0035】 I I . ビルディングブロックヘッダファイル

【0036】

【外2】

```

#include <ace/Service_Object.h>
#include <CsaConnectable.h>
#include <CsaRemote.h>
#include <SampleClass1.h>
#include <SampleClass2.h>
class SampleApplication : public ACE_Service_Object
{
public:
    virtual int init (int, char**);
    virtual int fini (void);
    virtual int info (char**, size_t) const;
    SampleApplication (void);
    ~SampleApplication (void);

protected:
    CsaConnectable <SampleClass1> *output1;
    CsaConnectable <SampleClass2> *output2;
    CsaRemote <SampleClass1> *input1;
    CsaRemote <SampleClass2> *input2;
};
#endif // SAMPLE_APPLICATION

```

【0037】 I I I . ビルディングブロック実施

【0038】

【外3】

```

#include<CsaConnectable.h>
#include<CsaRemote.h>
#include<SampleApplication.h>
int SampleApplication::init(int argc, char **argv) {
    cout << endl << "Initializing " << endl;
    input1 = new CsaRemote<SampleClass>(argv[1]);
    input2 = new CsaRemote<SampleClass>(argv[2]);
    output1 = new CsaConnectable<SampleClass>(argv[3]);
    output2 = new CsaConnectable<SampleClass>(argv[4]);
    return (0);
}
int SampleApplication::fini(void) {
    cout << endl << "Finalizing " << endl << endl;
    delete input1;
    delete input2;
    delete output1;
    delete output2;
    return (0);
}
int SampleApplication::info(char*, unsigned) const {
    cout << endl << "Returning info about " << endl;
    return (0);
}
SampleApplication::SampleApplication(void) {}
SampleApplication::~SampleApplication(void) {}
/* Dynamically linked functions used to control configuration */
extern "C" ACE_Service_Object *_alloc(void);
ACE_Service_Object *_alloc(void) {
    return (ACE_Service_Object *)new SampleApplication;
}

static SVC_Manager "d -p 3333"
dynamic SampleApplication
ACE_Service_Object *_alloc(void)
"SampleApplication in1_name in2_name out1_name out2_name"

```

【0039】IV. ASCII構築ファイル
 【0040】
 【外4】

【0041】図2には、ソフトウェアICを1つ以上のアプリケーションシステムで実施するブロック回路図が示されている。図2には5つのソフトウェアICが示されている。すなわち、IC1、IC2、IC3、IC4とIC5である。さらにソフトウェアICを使用する2つのアプリケーション、アプリケーション1とアプリケーション2が示されている。アプリケーション1はソフトウェアIC、IC1、IC2、IC3を含み、アプリケーション2はソフトウェアIC、IC4とIC5を含む。図からわかるように、アプリケーション1とアプリケーション2とは相互作用する。アプリケーション1とアプリケーション2とを含む外部プロセスまたは外部システムも同じようにソフトウェアICの人力と出力を介する。

【0042】図示のようにIC1は2つの人力側C11とC12を有する。IC1はまたR11を介する1つの出力側を有する。人力側C11とC12はIC2の2つの出力側、R21とR22にそれぞれ接続されている。IC2の人力側C21はIC1の出力側R11と接続されている。

【0043】IC3は出力側R31を有し、この出力側はIC2の人力側C22と接続されている。またIC3の人力側C31はアプリケーションを含む外部プロセスと接続されており、人力側C32はIC4の出力側R41と接続され、出力側R32はIC5の人力側C52および外部システムと接続されている。出力側R41に加えて、IC4は外部システムに接続された人力側C41と、IC5の人力側C51に接続された出力側R42を

有する。IC5はまた、アプリケーションを含む外部プロセスまたは外部システムと接続された出力側R51を有する。

【0044】入力側と出力側は、上に述べたようにCsaConnectableとCsaRemoteを介する。さらに、データが種々の入力側と出力側にダイナミックリンクを介してオートルーティングされる。これにより、コンフィギュレーションを変化することができ、アプリケーションが再コンパイルまたは再リンクをなしで相互作用することができる。

【0045】付加的に、前述のソフトウェアIC原理はACEからのパターン（タスク）と結合することができる。これは、強力なソフトウェアビルディングブロックを得るためであり、このソフトウェアビルディングブロックはハードウェアPALのような特性を有し、ビルディングブロック内での同期特性とビルディングブロックの外側での非同期特性／相互作用にパワーを与える。

【0046】内部処理率（ハードウェアPALでのクロック率に対するカウンタ部分）は従って十分に、接続された環境のイベント入／出力率から独立している。同期を達成するために必要なバッファはまた関連する意味構造なしで与えられる。ハードウェアPALの同期解決と同じように、同期タスクは必要であればソフトウェアPALに格納することができる。

【0047】図3は、ハードウェアPALとソフトウェアPALとの比較を示す。図示のように、ハードウェアPAL310はハードウェアICと同じように、2つの入力ピンI1とI2および2つの出力ピンO1とO2を有する。しかしハードウェアPAL310内にはまたレジスタ／バッファregが設けられており、ここに到来するデータおよび出力されるデータストアされる。

【0048】カウンタ部分ソフトウェアPAL312は、前に述べたソフトウェアICと同じように入力側R1、R2および出力側C1、C2を有する。しかしタスクT1とT2が図示されており、これらはハードウェア

PAL310のレジスタ／バッファregに代わるものである。他の点ではソフトウェアPALは上述のソフトウェアICと同じである。

【0049】ソフトウェアPALは、アクティブオブジェクトサポートによってソフトウェアICに対するフレキシビリティを内部ロジックに与える。到来するイベントはタスク、例えばタスクT1によって、内部ロジックによりさらに処理する前にバッファすることができる。これによりイベントはソフトウェアPALの内部ロジックから分離される。

【0050】当業者であれば変形および変更は可能であるが、それらも本願技術に含まれるものである。

【図面の簡単な説明】

【図1】ハードウェアICとソフトウェアICの比較を示す概略図である。

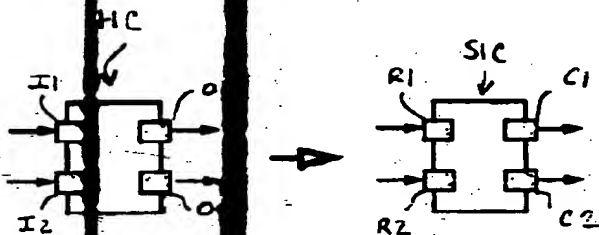
【図2】ソフトウェアICを使用したアプリケーションの概略図である。

【図3】ハードウェアPALとソフトウェアPALとの比較を示す概略図である。

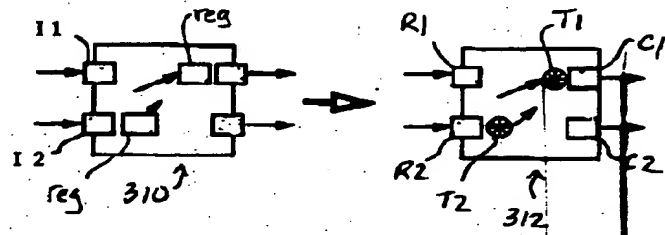
【符号の説明】

HIC ハードウェア集積回路（IC）
 I1 入力ピン
 I2 入力ピン
 O1 出力ピン
 O2 出力ピン
 SIC ソフトウェア集積回路（IC）
 R1 入力CsaRemote
 R2 入力CsaRemote
 C1 出力CsaConnectable
 C2 出力CsaConnectable
 IC1～IC5 ソフトウェアIC
 310 ハードウェアPAL
 312 ソフトウェアPAL
 T1、T2 タスク

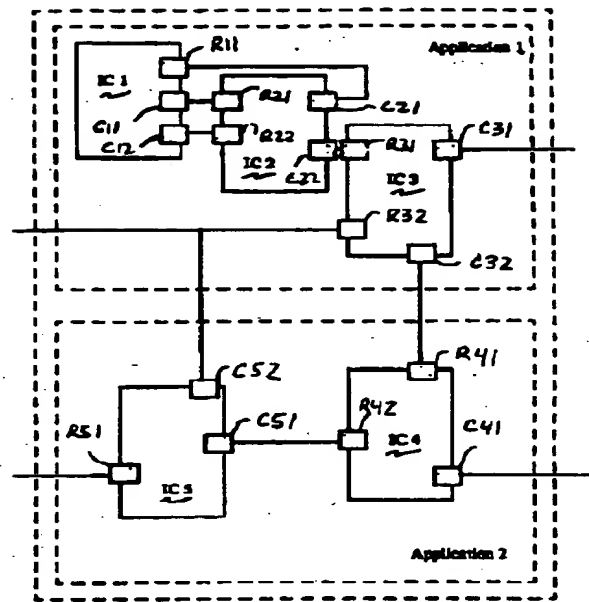
【図1】



【図3】



【図2】



フロントページの絵

(72)発明者 オトレ ベッカー
 ドイツ連邦共和国 メーレンドルフ ヴァ
 ヴァサー エルクシュトラッセ 10

(72)発明者 ディートリッヒ クエール
 ドイツ連邦共和国 エアランゲン ニュル
 ンベルガー シュトラッセ 83

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☒ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☒ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☒ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☒ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.